Hidden Markov Model
Latest Revision: 3/20/2006

Russell Loane

Notation:

POS means **Part Of Speech**.

In this doc, likelihood is used to describe an unnormalized probability. The sum of all likelihoods may not total to one. Likelihoods are used to compare the relative probabilities of alternatives, but not their absolute probabilities.

I was running into problems with subscripts on subscripts on subscripts. This problem arises most often with array elements, such as a_{ij}, which uses subscripts to indicate row and column. To avoid this problem, arrays are shown with a bracket notation, as in a[i,j], and the i and j are called indexes instead of subscripts.

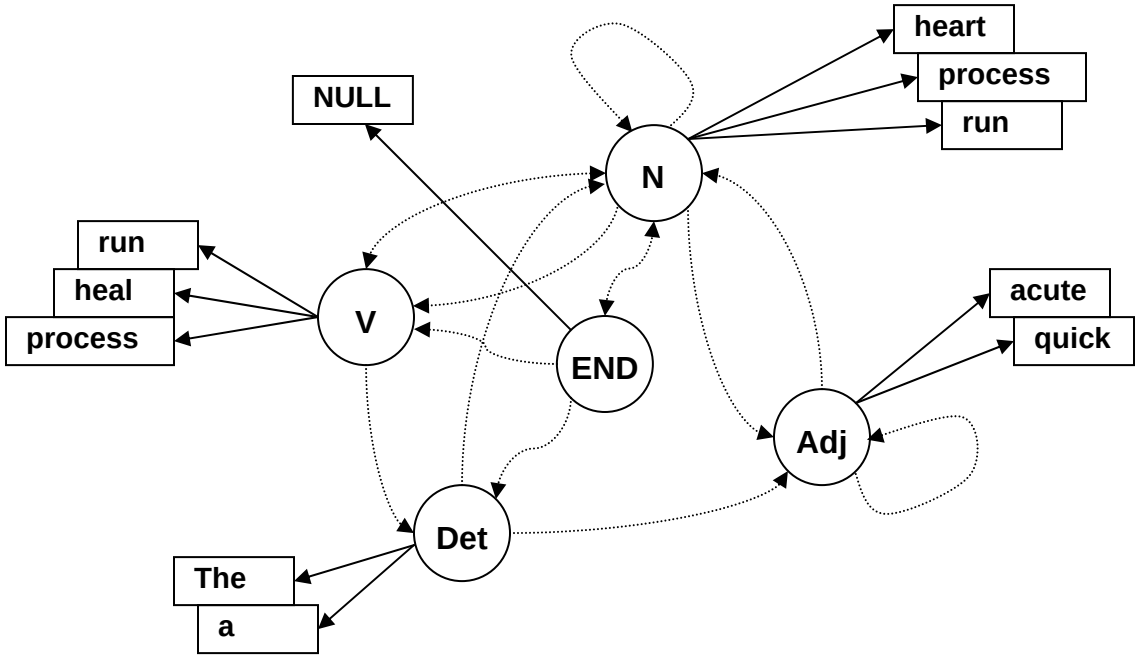
The Model:

A Markov Model (MM) consists of a set of states with probabilities of transitions between them.

In a Hidden Markov Model (HMM), the states have probabilities of emitting observations. When the same observation can be emitted by multiple states, it becomes ambiguous which state produced the observation. This problem never goes away, but the ambiguity can be quantified. The HMM makes it is possible to calculate the probability that a given state emitted a given observation.

For POS tagging, the states represent POS tags and the observations are words. Any given sentence is considered a sequence of observations. For each sentence, there are many possible sequences of POS tags (also called paths through state space). The HMM formalism allows us to calculate the probabilities that a given sequence is correct. We can also find the most likely, or best, POS sequence.

The figure is a simplified example of a HMM model for POS tagging. Circles are POS states. Rectangles are observed words. Arcs and arrows have a probability that they will be traversed. Note that the words "run" and "process" are emitted by two states.



Definitions:

Indexes start with zero so that the formula match the java implementation.
States are POS tags and are labeled with indexes i and j.
Observations are words and are labeled with index, n.
Positions of words and POS tags in a sentence are labeled with index, t.
Sentences in a corpus are labeled with index, m.

There is one special state, END, which is the first and last state of every state sequence.
There is one special observation, NULL, which is the only observation emitted by END.

A sentence with T words is a sequence of observations, $N = \{ n_0, n_1, n_2, ..., n_{T-1}, n_T, n_{T+1} \}$ where n_1 through n_T are words and n_0 and n_{T+1} are NULL.
A sentence with T words has a sequence of states, $I = \{ i_0, i_1, i_2, i_3, ..., i_{T-1}, i_T, i_{T+1} \}$ where i_1 through i_T are POSs and i_0 and i_{T+1} are END.
The first & last, word & state are sometimes omitted in discussion since they are fixed.

State transitions, a[i,j], are the probability of transition from state i (POS for current word) to state j (POS for next word) and are assumed to be independent of word position, t.
Observation emissions, b[i,n] are the probability of state i (POS for current word) producing output n (current word) and are assumed to be independent of word position, t.

Transitions and emissions are normalized:

Transitions occur: $\sum_j a[i,j] = 1$ but $\sum_i a[i,j]$ may not be 1 (1)

Emissions occur: $\sum_n b[i,n] = 1$ but $\sum_i b[i,n]$ may not be 1

Only ENDs emit NULLs: $b[END,n] = \delta_{NULL,n}$ and $b[i,NULL] = \delta_{i,END}$

Sequences:

For an arbitrary sentence of length T, the probability of state sequence $I = \{ i_0, i_1, i_2, i_3, ..., i_{T+1} \}$ occurring is the product of the transition probabilities:

$$P(I) = a[END,i_1] \cdot a[i_1,i_2] \cdot a[i_2,i_3] \cdot ... \cdot a[i_{T-1},i_T] \cdot a[i_T,END] \quad (2)$$

Note the restriction that state i_0 and i_{T+1} are always END.

For a given state sequence, I, the probability of an observation sequence $N = \{ n_0, n_1, n_2, ..., n_{T+1} \}$ occurring is the product of the emission probabilities corresponding to the states and words:

$$P(N|I) = 1 \cdot b[i_1,n_1] \cdot b[i_2,n_2] \cdot ... \cdot b[i_{T-1},n_{T-1}] \cdot b[i_T,n_T] \cdot 1 \quad (3)$$

Note that the probability of state END emitting NULL is 1.

Bayes theorem relates joint probabilities to conditional probabilities:

$$P(N,I) = P(N|I)\,P(I) = P(I|N)\,P(N) \quad (4)$$

So, the joint probability of I and N occurring is:

$$P(N,I) = a[END,i_1] \cdot b[i_1,n_1] \cdot a[i_1,i_2] \cdot b[i_2,n_2] \cdot a[i_2,i_3] \cdot ... \cdot a[i_{T-1},i_T] \cdot b[i_T,n_T] \cdot a[i_T,END] \quad (5)$$

The probability of a particular observation sequence occurring, P(N), is the sum of the joint probability over all possible state sequences, $I = \{ i_0, i_1, i_2, ..., i_{T+1} \}$:

$$P(N) = \sum_{i_1} \sum_{i_2} ... \sum_{i_T} a[END,i_1] \cdot b[i_1,n_1] \cdot a[i_1,i_2] \cdot ... \cdot a[i_{T-1},i_T] \cdot b[i_T,n_T] \cdot a[i_T,END] \quad (6)$$

By grouping terms one way,

$$P(N) = \sum_{i_T} \Big\{ \sum_{i_{T-1}} \Big\{ ... \Big\{ \sum_{i_2} \Big\{ \sum_{i_1} \Big\{ a[END,i_1] \cdot b[i_1,n_1] \Big\} \cdot a[i_1,i_2] \cdot b[i_2,n_2] \Big\} \cdot ... \Big\} \cdot a[i_{T-1},i_T] \cdot b[i_T,n_T] \Big\} \cdot a[i_T,END] \quad (7)$$

we can evaluate P(N) by forward induction:

$$\begin{aligned} f_1[i_1] &= a[END,i_1] \cdot b[i_1,n_1] \\ f_2[i_2] &= \sum_{i_1} f_1[i_1] \cdot a[i_1,i_2] \cdot b[i_2,n_2] \\ &... \\ f_T[i_T] &= \sum_{i_{T-1}} f_{T-1}[i_{T-1}] \cdot a[i_{T-1},i_T] \cdot b[i_T,n_T] \\ P(N) &= \sum_{i_T} f_T[i_T] \cdot a[i_T,END] \end{aligned} \quad (8)$$

This takes on the order of $T \cdot K^2$, where K is the number of states, floating point operations to evaluate.

Tagging:

In practice, we are given a particular sentence, N, and want to find the most likely POS sequence, I. Rearranging Bayes Theorem, eqn. (4), the probability of state sequence, I, given observation sequence, N, is:

$$P(I|N) = P(N,I) / P(N) \quad (9)$$

Note that P(N) is a constant for all I. Therefore, the most likely POS sequence that maximizes P(I|N) will also maximize P(N,I). We can maximize P(N,I) to find the most likely POS sequence. From eqn. (5) above:

$$P(N,I) = a[END,i_1] \cdot b[i_1,n_1] \cdot a[i_1,i_2] \cdot b[i_2,n_2] \cdot a[i_2,i_3] \cdot ... \cdot a[i_{T-1},i_T] \cdot b[i_T,n_T] \cdot a[i_T,END] \quad (10)$$

The best state sequence has T states which are not set to END, { i_1, i_2, i_3, ..., i_T }. We need to choose the values for these states to maximize eqn. (10).

So, for a particular i_2, i_3, ..., i_T, what is the best choice of i_1? The only factors that contain i_1 are:

$$a[END,i_1] \cdot b[i_1,n_1] \cdot a[i_1,i_2] \quad (11)$$

From this we see that the best choice of i_1 only depends on i_2 and not i_3, i_4, ..., i_T. So, for each i_2 find the best i_1, which we'll call $j_1[i_2]$.

$$j_1[i_2] = \text{the } i_1 \text{ which maximizes } a[END,i_1] \cdot b[i_1,n_1] \cdot a[i_1,i_2] \text{ for a given value of } i_2 \quad (12)$$

Thus, we have put off the decision of what is the best i_1 until we know the best i_2. It will also turn out to be useful to keep the maximum value, which we'll call $p_1[i_2]$.

$$p_1[i_2] = \text{the maximum value} = a[END,j_1[i_2]] \cdot b[j_1[i_2],n_1] \cdot a[j_1[i_2],i_2] \quad (13)$$

Moving on, for a particular i_3, i_4, ..., i_T, what are the best choices of i_1 and i_2? The only factors that contain i_1 and i_2 are:

$$a[END,i_1] \cdot b[i_1,n_1] \cdot a[i_1,i_2] \cdot b[i_2,n_2] \cdot a[i_2,i_3] \quad (14)$$

But, we already know the best i_1 given i_2 is $j_1[i_2]$, so the factors become:

$$a[END,j_1[i_2]] \cdot b[j_1[i_2],n_1] \cdot a[j_1[i_2],i_2] \cdot b[i_2,n_2] \cdot a[i_2,i_3] \quad (15)$$

which simplifies to:

$$p_1[i_2] \cdot b[i_2,n_2] \cdot a[i_2,i_3] \quad (16)$$

Like before, the best choice of i_2 only depends on i_3 and not i_4, ..., i_T. So, for each i_3 find the best i_2, which we'll call $j_2[i_3]$, and keep the maximum value, which we'll call $p_2[i_3]$.

$$j_2[i_3] = \text{the } i_2 \text{ which maximizes } p_1[i_2] \cdot b[i_2,n_2] \cdot a[i_2,i_3]$$
$$p_2[i_3] = \text{the maximum value} = p_1[j_2[i_3]] \cdot b[j_2[i_3],n_2] \cdot a[j_2[i_3],i_3] \quad (17)$$

Again, we have put off the decision of what is the best i_2 until we know the best i_3. Continuing on by induction:

$$j_3[i_4] = \text{the } i_3 \text{ which maximizes } p_2[i_3] \cdot b[i_3,n_3] \cdot a[i_3,i_4]$$

$$
\begin{aligned}
p_3[i_4] &= \text{the maximum value} = p_2[j_3[i_4]] \cdot b[j_3[i_4],n_3] \cdot a[j_3[i_4],i_4] \\
&\ldots \\
j_{T-1}[i_T] &= \text{the } i_{T-1} \text{ which maximizes } p_{T-2}[i_{T-1}] \cdot b[i_{T-1},n_{T-1}] \cdot a[i_{T-1},i_T] \\
p_{T-1}[i_T] &= \text{the maximum value} = p_{T-2}[j_{T-1}[i_T]] \cdot b[j_{T-1}[i_T],n_{T-1}] \cdot a[j_{T-1}[i_T],i_T]
\end{aligned}
\tag{18}
$$

The final choice of i_T does not depend on any subsequent states. So, try them all and keep the best:

$$
\begin{aligned}
j_T &= \text{the } i_T \text{ which maximizes } p_{T-1}[i_T] \cdot b[i_T,n_T] \cdot a[i_T,\text{END}] \\
p_T &= \text{the maximum value} = p_{T-1}[j_T] \cdot b[j_T,n_T] \cdot a[j_T,\text{END}]
\end{aligned}
\tag{19}
$$

This choice maximizes the likelihood of the entire sentence. Working backwards, the choices that for all of the most likely states are:

$$
\begin{aligned}
&j_T \\
&j_{T-1}[j_T] \\
&j_{T-2}[j_{T-1}[j_T]] \\
&j_{T-3}[j_{T-2}[j_{T-1}[j_T]]] \\
&\ldots \\
&\\
&I_{best} = \{ j_1, j_2, j_3, \ldots, j_T \}
\end{aligned}
\tag{20}
$$

where I_{best} is the most likely sequence of POS that match the sentence. This is the **Viterbi** algorithm and takes on the order of $T \cdot K$, where K is the number of states, floating point operations to evaluate.

A rearrangement improves computational efficiency slightly:

$$
\begin{aligned}
q_0[i_1] &= a[\text{END},i_1] \cdot b[i_1,n_1] \\
\\
j_1[i_2] &= \text{the } i_1 \text{ which maximizes } q_0[i_1] \cdot a[i_1,i_2] \\
q_1[i_2] &= q_0[j_1[i_2]] \cdot a[j_1[i_2],i_2] \cdot b[i_2,n_2] \\
\\
j_2[i_3] &= \text{the } i_2 \text{ which maximizes } q_1[i_2] \cdot a[i_2,i_3] \\
q_2[i_3] &= q_1[j_2[i_3]] \cdot a[j_2[i_3],i_3] \cdot b[i_3,n_3] \\
\\
&\ldots \\
\\
j_{T-1}[i_T] &= \text{the } i_{T-1} \text{ which maximizes } q_{T-2}[i_{T-1}] \cdot a[i_{T-1},i_T] \\
q_{T-1}[i_T] &= q_{T-2}[j_{T-1}[i_T]] \cdot a[j_{T-1}[i_T],i_T] \cdot b[i_T,n_T] \\
\\
j_T &= \text{the } i_T \text{ which maximizes } q_{T-1}[i_T] \cdot a[i_T,\text{END}] \\
q_T &= q_{T-1}[j_T] \cdot a[j_T,\text{END}]
\end{aligned}
$$

Note that the final sequence probabilities are the same, $p_T = q_T$.

Path Probabilities:

The maximum value, p_T, determined in eqn. (19) is the joint probability, $P(N,I_{Best})$. for the most likely path, I_{Best}. If we evaluate P(N) from eqn (8), we can use eqn (9) to determine the probability of this best sequence of POSs occurring, P_{Best}, given the observed sequence words, N.

$$P_{Best} = P(I_{Best}|N) = p_T / P(N) \quad (21)$$

For a straight-forward, unambiguous sentence, this probability should be high, $P_{Best}>0.5$. If it is ambiguous whether a word is a noun or an adjective, then there are two or more likely paths, and the probability for the best sequence of POSs will drop.

Note that once P(N) is known, you can derive the probability of any path with eqns. (9) & (10). One thing to try is to take the best path, make small variations, and evaluate the probabilities for the nearby alternative paths.

State Probabilities:

Please review eqns. (6-8). Recall that deriving the probability of an observation sequence, P(N), resulted in the first eqn. in (8):

$$f_1[i_1] = a[END,i_1] \cdot b[i_1,n_1] \quad (22)$$

If i_1 and n_1 are unconstrained, the right hand side is the probability of transitioning to state i_1 and emitting a word n_1. But for a given sentence, we already know what n_1 is. If we only consider the cases for a particular n_1, the f_1s do not add up to one. So, the f_1s are likelihoods of transitioning to the first state, i_1, given the first word, n_1. Normalize to get back to probabilities.

With normalization, we get:

$$F_1[i_1] = f_1[i_1] / \sum i_1 \; f_1[i_1] \quad (23)$$
$$\sum i_1 \; F_1[i_1] = 1$$

We interpret $F_1[i_1]$ as the probability that a given first word, n_1, is in state, i_1.

The same logic lets us interpret $F_t[i_t]$ as the probability that the word, n_t, is in state, i_t, given all words before (and including) position t.

$$F_t[i_t] = f_t[i_t] / \sum i_t \; f_t[i_t] \quad (24)$$

Again, $f_t[i_t]$ is the corresponding likelihood. This applies for any position, t, in the sentence.

We can use a similar approach to find the probability that a word, n_t, is in state, i_t, given all words after t. By grouping terms in eqn. (6) another way:

$$P(N) = \sum i_1 \left\{ \sum i_2 \left\{ \ldots \left\{ \sum i_{T-1} \left\{ \sum i_T \left\{ a[i_T,END] \right\} \cdot a[i_{T-1},i_T] \cdot b[i_T,n_T] \right\} \cdot \ldots \right\} \cdot a[i_1,i_2] \cdot b[i_2,n_2] \right\} \cdot a[END,i_1] \cdot b[i_1,n_1] \quad (25)$$

P(N) can be evaluated by backward induction:

$$g_T[i_T] = a[i_T,END]$$
$$g_{T-1}[i_{T-1}] = \sum i_T \; a[i_{T-1},i_T] \cdot b[i_T,n_T] \cdot g_T[i_T]$$

...

$$g_1[i_1] = \sum i_2 \; a[i_1,i_2] \cdot b[i_2,n_2] \cdot g_2[i_2] \tag{26}$$

$$P(N) = \sum i_1 \; a[END,i_1] \cdot b[i_1,n_1] \cdot g_1[i_1]$$

We interpret $g_t[i_t]$ as the likelihood that the word, n_t, is in state, i_t, given all words after t.

With some additional regrouping of eqn. (6), both approaches can be combined for any t:

$$P(N) = \sum i_t \; f_t[i_t] \cdot g_t[i_t] \tag{27}$$

The product, $f_t[i_t] \cdot g_t[i_t]$, is the likelihood that the word, n_t, is in state, i_t, given all the words in the sentence. Normalize to get back to probabilities:

$$S_t[i_t] = f_t[i_t] \cdot g_t[i_t] / \sum i_t \; f_t[i_t] \cdot g_t[i_t] = f_t[i_t] \cdot g_t[i_t] / P(N) \tag{28}$$

$$\sum i \; S_t[i_t] = 1$$

$S_t[i_t]$ is the probability that the word, n_t, is in state, i_t, given all the words in the sentence. This is essentially a normalized sum over all paths that go through state, i_t.

For a straight-forward, unambiguous sentence, each word, n_t, should have only one likely POS, i_t. If more than one state has a significant probability, then the corresponding word has an ambiguous POS. You can use these state probabilities to generate nearby alternatives to the best path.

It is possible to calculate, I_{best}, P_{Best}, P(N), $f_t[i_t]$, $g_t[i_t]$, and $S_t[i_t]$ with a reasonable amount of computation. Given these values, it is easy to explore the best POS fit to a sentence, and all nearby variants. In particular, when a fit is poor, it is possible to see where the ambiguity is, and decide whether the model is doing a poor job or the sentence is inherently ambiguous.

Supervised Training with a Graded Corpus:

Given a tagged corpus, the values for a[i,j], and b[i,n] can be estimated.

Define the A[i,j] as the transition counts. Each pair of adjacent words, n_t and n_{t+1} correspond to a transition, i_t to i_{t+1}. Increment $A[i_t,i_{t+1}]$. Do the same thing for the END transitions at sentence starts and sentence ends. Transition probs, a[i,j] are the A[i,j] normalized for each i with eqn (1).

Define the B[i,n] as the emission counts. Each occurrence of a word, n_t, corresponds to an emission for a POS, i_t. Increment $B[i_t,n_t]$. Emission probs, b[i,n] are the B[i,n] normalized for each i with eqn (1).

There needs to be some slop in this to account for an incomplete graded corpus. This will, of course, add noise. I'm arbitrarily adding a (very) small chance that any POS state, i, will transition to any state, j, and emit any word, n. In each POS, i, 1 count is split amongst all js in a[i,j] and all the ns in b[i,n] (except END still only emits NULL by definition). Normalization is performed with eqn (1).

Unknown Words:

Conceptually, there exists a b[i,n] for every word n. In reality, there are too many words to list them all. Even if you could, new words are created all the time, so the problem remains. With a good model, all the common words are already known and in the model. Only odd uncommon words remain unknown. It turns out that it is a bad approximation to extrapolate the behavior of the known words to the unknowns -- they are qualitatively different.

Imagine a model with 50 thousand nouns trained on a corpus with 100 thousand sentences and 1 million words. Of those nouns, about half will have only a single occurrence. If we shrink our training corpus by 1 carefully selected sentence, we can turn a singly occurring noun into an unknown word. Comparing the models for the original and shrunk training sets, we can see the correct way to handle unknown words. This trick can be played with all singly occurring words in any part of speech.

Lets create an emission prob for unknown words. Conceptually, this is treated as slot b[i,N] in the existing emission probs. In the implementation, I keep this value separate in an "unknown" emission prob, $b_?[i]$. Assuming that the training corpus is big enough, adding the next sentence to the corpus should be the same as adding the previous sentence. The chance that the previous sentence added a new word is the ratio of (the number of occurrences for words with a single occurrence) to (the total number of occurrences). So,

$$b_?[i] = \sum_n \delta B[i,n] \; / \; \sum_n B[i,n]$$

where

$$\delta B[i,n] = \begin{cases} 1 & \text{if } B[i,n] = 1, \\ 0 & \text{otherwise} \end{cases}$$

If X% of the counts fall into the unknown emission prob, then the emission probs for all the known words should normalize to (100-X)%. This is just the familiar normalization from eqn. 1, with one slot treated separately.

Using the Lexicon:

There are thousands of times as many different words as there are parts of speech. So, there are far more b[i,n] than a[i,j]. We will never have a large enough training set to get good estimates of all the b[i,n]. But, we can use the Specialist Lexicon to create crude estimates. If the lexicon states word n cannot occur in POS, i, we decrease b[i,n] or $b_?[i]$ by some constant factor, currently set to 3.0.

The suppression can be included in the b[i,n] for known words when the model is built.
The suppression can be applied to the $b_?[i]$ when an unknown word is considered.

Unsupervised Training with an Ungraded Corpus:

Valid sentences are a small structured subset of the set of all word sequences. Assuming that some of this structure can be captured in a HMM model, it makes sense to adjust the model to increase the probability of emitting a set of known sentences. Of course, it is possible to increase the probabilities, even if the model does

not capture the essence of sentence structure. In this case, a model trained on one set of sentences will not work well with a different set.

The general strategy is 1) apply the current model to a corpus of sentences, 2) determine the actual transition, emission, and initial probabilities that occurred with the corpus, and 3) replace the original model probabilities with the actual probabilities. The hope here is that if the current model is slightly wrong, the set of real sentences will force a different behavior which can then be captured.

This clearly breaks down when one of the transition probabilities is zero. In this case, it is impossible for real sentences to force the probability to be non-zero. This is a problem even if the original model has no zero transition probabilities in it. A non-representative training set of sentences may not include any examples of a particular transition or word, and therefore drive those probabilities to zero. There needs to be some slop in the training to account for an incomplete corpus. Some experimentation is required to figure out what is the best way to handle this.

I find the rationale for this approach to be weak. It seems absurd to think you can do a good job of optimizing a million non-linear parameters simultaneously. But, people have had some success with this in the past. We're going to try it and see.

Now we derive the equations to update the HMM. Recall from eqn. (27):

$$P(N) = \sum_{i_t} f_t[i_t] \cdot g_t[i_t] \tag{31}$$

The product, $f_t[i_t] \cdot g_t[i_t]$, is the likelihood that the word, n_t, is in state, i_t, given all the words in the sentence. Expanding out one step of recursion for $g_t[i_t]$ and yields:

$$P(N) = \sum_{i_t} \sum_{i_{t+1}} f_t[i_t] \cdot a[i_t,i_{t+1}] \cdot b[i_{t+1},n_{t+1}] \cdot g_{t+1}[i_{t+1}] \tag{32}$$

We interpret the product, $f_t[i_t] \cdot a[i_t,i_{t+1}] \cdot b[i_{t+1},n_{t+1}] \cdot g_{t+1}[i_{t+1}]$, as the likelihood that the words, n_t & n_{t+1}, are in states, i_t & i_{t+1}, given all the words in the sentence. Normalizing, yields probabilities:

$$T_t[i_t,i_{t+1}] = f_t[i_t] \cdot a[i_t,i_{t+1}] \cdot b[i_{t+1},n_{t+1}$$

where $\delta[n_{t+1},n]$ is the Kroneker delta function which is 1 for $n_t = n$ and zero everywhere else. The B[i,n] are counts of emissions observed when the model is applied to a corpus. A new estimate for the emission probability, b[i,n] are these values, B[i,n], normalized for each i with eqn (1).

Rescaling:

Since there are so many words, the emission probabilities, b[i,n], are generally small. For long sentences, this can cause underflow. We can solve this problem by scaling the emission probabilities to make them bigger:

$$w[n] = \sum i \; b[i,n]$$
$$bb[i,n] = b[i,n] / w[n] = b[i,n] / \sum i \; b[i,n]$$
$$b[i,n] = bb[i,n] \cdot w[n] = bb[i,n] \cdot \sum i \; b[i,n]$$
$$\sum i \; bb[i,n] = 1$$

It is tempting to think of bb[i,n] as the probability that an instance of word n is in POS state i. This is what I did in the old model and it just plain **WRONG!** The emission probabilities are not a 2D probability distribution. They are a set of 1D distributions, with one for each POS, i. There is no simple relationship between b[i,n] for different values if i. By converting to bb[i,n], we are only factoring out a scaling factor, w[n]. We are not renormalizing likelihoods to probabilities. The b[i,n] for a given n are not likelihoods. This still trips me up from time to time.

Now convert all of the equations from b[i,n] to bb[i,n]. Scaled versions of important quantities are indicated with a double letter name.

$$W(N) = \left\{ w[n_1] \cdot w[n_2] \cdot ... \cdot w[n_{T-1}] \cdot w[n_T] \right\}$$

$$PP(N|I) = P(N|I) / W(N) = \left\{ bb[i_1,n_1] \cdot bb[i_2,n_2] \cdot ... \cdot bb[i_{T-1},n_{T-1}] \cdot bb[i_T,n_T] \right\}$$

$$PP(N,I) = P(N,I) / W(N)$$
$$= a[END,i_1] \cdot bb[i_1,n_1] \cdot a[i_1,i_2] \cdot bb[i_2,n_2] \cdot a[i_2,i_3] \cdot ... \cdot a[i_{T-1},i_T] \cdot bb[i_T,n_T] \cdot a[i_T,END]$$

$$PP(N) = P(N) / W(N)$$
$$= \sum i_1 \; \sum i_2 \; ... \; \sum i_T \; a[END,i_1] \cdot bb[i_1,n_1] \cdot a[i_1,i_2] \cdot ... \cdot a[i_{T-1},i_T] \cdot bb[i_T,n_T] \cdot a[i_T,END]$$

The scaled forward likelihoods:

$$ff_1[i_1] = a[END,i_1] \cdot bb[i_1,n_1]$$
$$ff_2[i_2] = \sum i_1 \; ff_1[i_1] \cdot a[i_1,i_2] \cdot bb[i_2,n_2]$$
...
$$ff_T[i_T] = \sum i_{T-1} \; ff_{T-1}[i_{T-1}] \cdot a[i_{T-1},i_T] \cdot bb[i_T,n_T]$$
$$PP(N) = \sum i_T \; ff_T[i_T] \cdot a[i_T,END]$$

The scaled backward likelihoods:

$gg_T[i_T] = a[i_T,END]$

$gg_{T-1}[i_{T-1}] = \sum i_T \; a[i_{T-1},i_T] \cdot bb[i_T,n_T] \cdot gg_T[i_T]$

...

$gg_1[i_1] = \sum i_2 \; a[i_1,i_2] \cdot bb[i_2,n_2] \cdot gg_2[i_2]$

$PP(N) = \sum i_1 \; a[END,i_1] \cdot bb[i_1,n_1] \cdot gg_1[i_1]$

The state probabilities:

$PP(N) = \sum i_t \; ff_t[i_t] \cdot gg_t[i_t]$

$S_t[i_t] = ff_t[i_t] \cdot gg_t[i_t] / \sum i_t \; ff_t[i_t] \cdot gg_t[i_t] = ff_t[i_t] \cdot gg_t[i_t] / PP(N)$

$\sum i \; S_t[i_t] = 1$

The Viterbi algorithm:

$qq_0[i_1] = a[END,i_1] \cdot bb[i_1,n_1]$

$j_1[i_2]$ = the i_1 which maximizes $qq_0[i_1] \cdot a[i_1,i_2]$

$qq_1[i_2] = qq_0[j_1[i_2]] \cdot a[j_1[i_2],i_2] \cdot bb[i_2,n_2]$

$j_2[i_3]$ = the i_2 which maximizes $qq_1[i_2] \cdot a[i_2,i_3]$

$qq_2[i_3] = qq_1[j_2[i_3]] \cdot a[j_2[i_3],i_3] \cdot bb[i_3,n_3]$

...

$j_{T-1}[i_T]$ = the i_{T-1} which maximizes $qq_{T-2}[i_{T-1}] \cdot a[i_{T-1},i_T]$

$qq_{T-1}[i_T] = qq_{T-2}[j_{T-1}[i_T]] \cdot a[j_{T-1}[i_T],i_T] \cdot b[i_T,n_T]$

j_T = the i_T which maximizes $qq_{T-1}[i_T] \cdot a[i_T,END]$

$qq_T = qq_{T-1}[j_T] \cdot a[j_T,END]$

$pp_T = qq_T$

Fraction of sentence probability in best path:

$p_T / P(N) = pp_T / PP(N)$

Update counts:

$<A[i,j]> = \sum m \; \sum t \left\{ ff_t[i] \cdot a[i,j] \cdot bb[j,n_{t+1}] \cdot gg_{t+1}[j] / PP(N) \right\}$

$<B[i,n]> = \sum m \; \sum t \; \delta[n_t,n] \cdot S_t[i_t]$

As shown, the inclusion of a scaling factor does not change the form of the equations. The scaling factors all cancel out in the final results. It just doesn't matter. In the code, we use the single letter names for rescaled quantities.